E190AR: Lab 2 Report

Charles Dawson & Peter Johnson

***Abstract*— This report describes the development of a ROS node to address both forward kinematics and odometry to allow velocity control of the robot, as well as its localization based on wheel encoders. Experimental results from motor calibration, odometry testing, and IR range sensor calibration are also presented, along with discussions of odometry accuracy and IR range sensor accuracy, precision, and nonlinearity.**

I. Introduction

In order to get where you're going, you need to know both where you've been and how to move forward. Forward kinematics allows us to move forward by determining what motor control inputs will produce the desired robot motion. On the other hand, odometry allows us to know where we've been by using measurements from rotary encoders on the robot's wheels to continuously update the location and orientation of the robot in the global frame.

In this lab, we integrate forward kinematics and odometry into a single ROS node that subscribes to `/cmd_vel` (the desired local robot velocities in the local frame) and publishes `/odom` (the continuously-updated robot location and orientation in the global frame). This control node provides the framework for future explorations of advanced concepts in robotics, as it provides low-level motor control and position monitoring for our robot, which an be later integrated with closed-loop motion planning and maneuver execution strategies.

II. Theory

This section presents the mathematical theory for forward kinematics and wheel odometry for the differential drive E190 robot shown in Fig. 1.

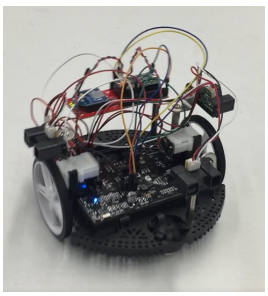

Fig. 1. Assembled Romi Pololu, the robotics platform used in E190AR.

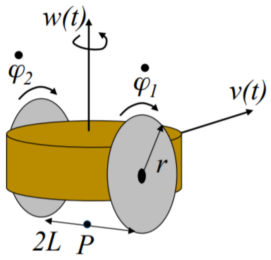

Fig. 2. Schematic of E190 robot, showing robot velocity and wheel velocity [1].

A. Forward Differential Drive Kinematics

The E190AR robot shown in Fig. 1 is under-actuated, with direct control over only linear velocity $v(t)$ in the local x direction and rotational velocity $w(t)$ in the local z direction, as shown in Fig. 2. In order to determine the relationship between velocities $v(t)$ and $w(t)$ and our control input (the speeds of the right and left wheels, $\dot{\phi}_1(t)$ and $\dot{\phi}_2(t)$ respectively), we observe, following the work presented in lecture [1], that the linear speed of a wheel of radius r rotating at angular speed $\dot{\phi}_i$ is

$$v_i(t) = r\dot{\phi}_i(t). \tag{1}$$

Thus, at any given point in time, the linear velocity of the robot in the x-direction, $v(t)$, must be the average of the linear velocities of the two wheels, so

$$v(t) = \frac{r}{2}\left[\dot{\phi}_1(t) + \dot{\phi}_2(t)\right] \tag{2}$$

To determine the relationship between $\dot{\phi}_1(t)$, $\dot{\phi}_2(t)$, and $w(t)$, we can consider the action of each wheel independently. If we set $\dot{\phi}_1(t) = 0$ (i.e. stopping the right wheel), the robot will rotate clockwise ($w(t) < 0$) about its right wheel as shown on the left in Fig. 3. Since the linear velocity at radius $2L$ from the center of rotation is $v_2 = r\dot{\phi}_2(t)$, we know that the angular velocity of this rotation must be $\omega_2 = -v_2/(2L)$, or

$$\omega_2 = -\frac{r\dot{\phi}_2(t)}{2L}. \tag{3}$$

Likewise, stopping the left wheel and observing the counter-clockwise rotation yields the relation

$$\omega_1 = \frac{r\dot{\phi}_1(t)}{2L}. \tag{4}$$

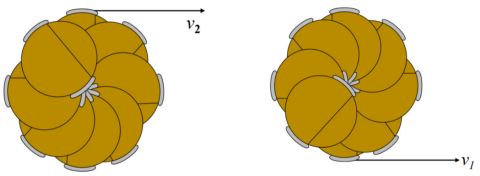

Fig. 3. Motion of the robot when only one wheel turns [1].

Linearly combining these two modes of rotational motion allows us to determine the angular velocity of the robot when both motors are spinning:

$$w(t) = \omega_1(t) + \omega_2(t) \tag{5}$$
$$w(t) = \frac{r}{2L}\left[\dot{\phi}_1(t) - \dot{\phi}_2(t)\right] \tag{6}$$

When controlling the robot, we would like to be able to specify v and w, then let the motion control node determine what wheel speeds $\dot{\phi}_1$ and $\dot{\phi}_2$ are needed to produce the desired linear and rotational velocities. To achieve this goal, we need to invert the kinematic equations 2 and 6. Combining these equations and solving for $\dot{\phi}_1$ and $\dot{\phi}_2$ yields

$$\dot{\phi}_1 = \frac{1}{r}\left[v(t) + Lw(t)\right] \tag{7}$$
$$\dot{\phi}_2 = \frac{1}{r}\left[v(t) - Lw(t)\right] \tag{8}$$

These equations, when combined with the calibration data discussed in Section III, will allow the motor control node to determine the left and right wheel speeds needed to accomplish arbitrary combinations of translation in the x-direction and rotation about the z-axis.

B. Wheel Odometry

To provide the foundation for later closed-loop robot controllers, we need to ensure that the robot is able to estimate its current location (or at least its current location relative to some start point). To do this, we will use magnetic encoders to measure the angular position of the wheels over time, then use those measurements to continuously update the position of the robot in the global fixed frame.

The Teensy microcontroller on our robot sends encoder measurements to the control node via the XBee at a rate of 2 Hz. The encoders have a resolution of 1440 count/revolution, so the left and right encoder readings ($\Delta\text{Enc}_\text{L}$ and $\Delta\text{Enc}_\text{R}$, measured by subtracting the previous encoder reading from the current reading) can be converted to linear distances travelled by the the left and right wheel according to the formulas

$$\Delta s_R = \Delta\text{Enc}_\text{R}\frac{2\pi\text{r}}{1440} \tag{9}$$
$$\Delta s_L = \Delta\text{Enc}_\text{L}\frac{2\pi\text{r}}{1440} \tag{10}$$

From these wheel translations, we can compute the incremental translation Δs and rotation θ of the robot body, shown in Fig. 4, using the equations

$$\Delta s = \frac{\Delta s_r + \Delta s_L}{2} \tag{11}$$
$$\Delta\theta = \frac{\Delta s_r - \Delta s_L}{2L} \tag{12}$$
$$\tag{13}$$

As discussed in lecture [1], we can describe the state of the robot by its location, a point in the global xy frame $\mathbf{p} = [p_x, p_y, 0, 1]^T$, and its rotation, the angle θ between the global x-axis and the local robot x-axis. In a given state, the homogeneous transformation H from the local frame to the global frame is

$$H = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & p_x \\ \sin\theta & \cos\theta & 0 & p_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{14}$$

If the robot state is updated incrementally by rotating an angle $\Delta\theta$ and moving a distance Δs along the newly rotated x-axis (as shown in Fig. 4), we can describe the post-update robot location in the pre-update frame (i.e. the local coordinate frame before the robot state is updated) as

$$\mathbf{p}' = [\Delta s\cos(\Delta\theta/2), \Delta s\sin(\Delta\theta/2), 0, 1]^T \tag{15}$$

if we assume that the frequency at which the robot state is updates is sufficiently fast relative to the robot speed (so that we can approximate the incremental translation shown in Fig. 4 as a straight line).

Transforming $\mathbf{p}'$ into the global frame using the homogeneous transformation H yields the updated robot position in the global frame:

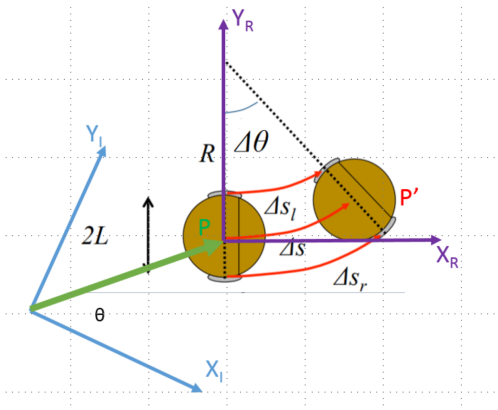

Fig. 4. Incremental wheel odometry update [1].

$$\mathbf{p}'^{(g)} = H\mathbf{p}' \tag{16}$$

$$\mathbf{p}'^{(g)} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & p_x \\ \sin\theta & \cos\theta & 0 & p_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{p}' \tag{17}$$

$$\mathbf{p}'^{(g)} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & p_x \\ \sin\theta & \cos\theta & 0 & p_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta s\cos(\Delta\theta/2) \\ \Delta s\sin(\Delta\theta/2) \\ 0 \\ 1 \end{bmatrix} \tag{18}$$

$$\mathbf{p}'^{(g)} = \begin{bmatrix} \Delta s\cos(\theta+\Delta\theta/2) + p_x \\ \Delta s\sin(\theta+\Delta\theta/2) + p_y \\ 0 \\ 1 \end{bmatrix} \tag{19}$$

Furthermore, the orientation of the robot frame must be updated as well, so $\theta' = \theta + \Delta\theta$.

III. Experimental Calibration

This section describes the procedures used to calibrate the motor control, wheel odometry, and IR distance sensors of our robot.

A. Open-loop Motor Control

In Section II, we derived the kinematic equations needed to determine the wheel speeds needed to achieve desired robot motion; however, we cannot control wheel speed directly. Since we can only control wheel speed by setting a PWM value between 0 and 255, we need to calibrate the relationship between the PWM control signal and the resulting wheel speed. To do this, we commanded the robot to drive both wheels forward at a constant PWM value for 10 seconds, then measured the difference between the starting and ending values of the left and right wheel encoders. These encoder differences can be divided by 1440 (the resolution of the encoders) and multiplied by $2\pi/10$ seconds to yield a measurement of the wheel speed in radians per second (which can be used in the kinematic equations from Section II). The results of this calibration are shown in Fig. 5. We observed fairly linear behavior for both motors at low motor speeds, so we decided to calibrate our motors using a linear model fit to the low-speed regime. Because the odometry calculations described in Section II rely on the assumption that the robot is moving slowly relative to its 2 Hz update rate, the robot odometry is already limited to low speeds, so we decided to make our calibration models most accurate in the same low-speed regime.

Based on these calibration curves, we derived the following linear model relating the desired wheel speed to command input:

$$\dot{\phi}_1 = k_R\frac{PWM_R}{255}; \quad \dot{\phi}_2 = k_L\frac{PWM_L}{255} \tag{20}$$

where $k_R = 31.0838$ and $k_L = 30.5738$. Inverting these equations yields the formulas for determining what PWM signal to output to achieve desired wheel speeds:

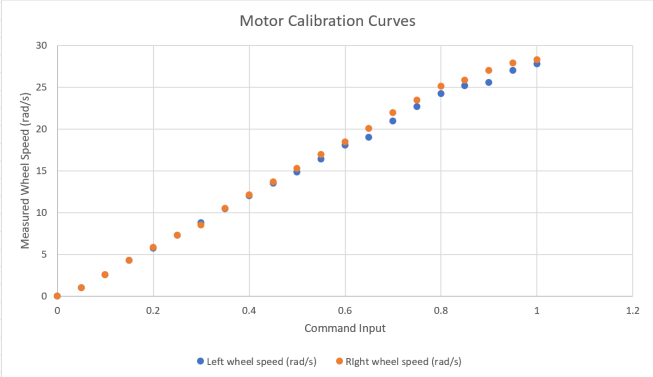

Fig. 5. Calibration curve for left and right motors, showing measured wheel speed in radians/second as a function of command input, where the PWM signal = command input $*$ 255.

$$PWM_R = \frac{255\dot{\phi}_1}{k_R} \tag{21}$$

$$PWM_L = \frac{255\dot{\phi}_2}{k_L}. \tag{22}$$

Combined with the kinematic equations in Section II, these formulae will allow us to implement a control node (discussed in Section IV) that will be able to achieve arbitrary combinations of linear x and rotational z motion.

B. Wheel Odometry

After implementing the odometry equations described in Section II, we tested the accuracy of our odometry system by instructing the robot to drive in a 1 meter square pattern with only open-loop control during the in-class demo. After it completed the maneuver, we measured the distance from the robot's starting position to its final position and compared that ground truth with the prediction in the published `/odom` topic. After completing the maneuver during the in-class demonstration, we measured a ground-truth position of $x = 13$ cm and $y = 21$ cm, while the odometry reported an estimated prediction of $x = 10$ cm and $y = 20$ cm. The distance between the estimated and ground-truth positions was $d_{\text{error}} = 3.16$ cm, or just 0.79% of the total distance driven during the maneuver (400 cm). Based on multiple successful tests, shown in Table. 6, we concluded that our odometry measurements were functioning well. Note that the results shown in Table 6 do not end up as close to the origin as in the class demo test because we calibrated our program to trace out a square running on carpet, and the tests in Table 6 were done on tile floor, so the robot ended up tracing out a shape more like a triangle; however, the odometry consistently estimated the state of the robot to within 6 cm, even after travelling 400 cm in total (with an average accumulated error of 5.1 cm, or 1.27% of total distance travelled).

Trial #	Ground-Truth (x, y)	Estimated (x, y)	Error
1	(1.10 0.14)	(1.13, 0.16)	3.6 cm (0.9%)
2	(1.00, 0.25)	(0.95, 0.28)	5.8 cm (1.45%)
3	(1.14, 0.15)	(1.09, 0.18)	5.8 cm (1.45%)
Avg	n/a	n/a	5.1 cm (1.27%)

Fig. 6. Tests of wheel odometry measurements. All locations are in meters, while error has units of centimeters.

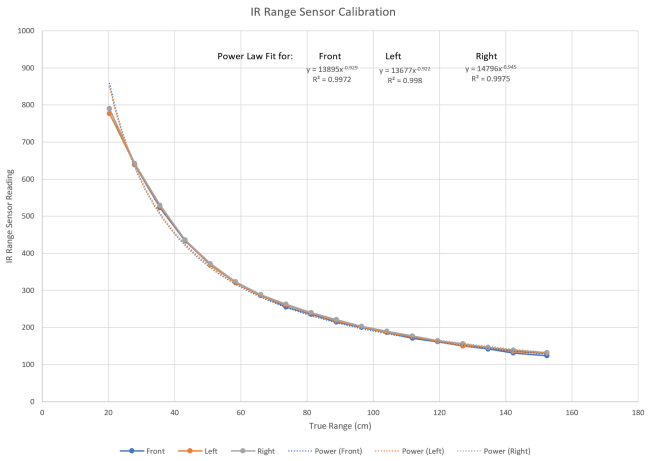

Fig. 7. Calibration curves for IR distance sensors, showing a high degree of nonlinearity in the relation between distance and output reading.

C. IR Sensors

The E190AR robot uses three SHARP GP2Y0A02YK0F IR range sensors to measure the distance from the robot to the nearest obstacle in the local $+x$, $+y$, and $-y$ directions. The datasheet for this sensor specifies that it has a useful range of 20-150 cm, so we calibrated each sensor over that same range [2]. Since the sensors output an analog voltage that our robot reads via an analog-to-digital converter with 10 bits of resolution, the recorded readings can theoretically range from 0 to 1024, but practically only values between 100-800 were observed. We calibrated each sensor by taking readings at 7.62 cm (3 in) increments from 20-150 cm. At each distance, we recorded 10000 samples from the IR sensor and averaged them to yield a single reading, along with a measurement of the variance (i.e. precision) of the sensor measurement. These data are shown in Fig. 7, and we observed very close agreement in the data from all three sensors.

Obviously, this sensor is extremely nonlinear (with a calculated linearity of 25%), but we found that all three sensors could be modelled fairly well using a power law, where

$$\text{Sensor Reading} = A(\text{Range})^B.$$

Because of the extremely close agreement between the three sensors, we decided to use a single power law to model all three sensors, choosing the parameters $A = 13895$ and $B = -0.929$ derived from a power law fit of the data from the front-facing sensor. We then inverted that model to find a formula for calculating range given a sensor reading:

$$\text{Range} = \left(\frac{\text{Sensor Reading}}{A}\right)^{1/B} \tag{23}$$

Using the parameters provided above, this model is very accurate. For ranges greater than 25 cm, we observed a maximum relative error (over all ranges from 25-150 cm) between ground-truth range and the range measured using our calibrated model of 6.51% for the front sensor, 3.11% for the left sensor, and 0.40 % for the right sensor (producing accuracies of 93.49%, 96.89%, and 99.60%, respectively). However, the sensor readings were fairly imprecise; we measured an standard deviation in sensor reading of ≈ 14 (out of 1024) across all of our sensor readings. This imprecision is least problematic at low ranges, where the calibration model is less sensitive to fluctuations in sensor reading, but this imprecision limits our ability to quickly obtain reliable range measurements at distances > 100 cm, where the standard deviation exceeds 10% of the sensor reading (this limitation can be overcome by averaging a large number of data points).

IV. Control Node Implementation

This section describes the implementation of the robot control node. The `/cmd_vel` topic is subscribed to by our control node and is how our robot receives kinematics commands. The `/cmd_vel` messages come in the form of a geometry_msg; Twist. This data type is used for all movement commands. The Twist is composed of two Vector3 objects a "Linear" and an "Angular" for both of these, the velocity is specified for the x, y, z directions of the robot (local or base_link) frame. In our case, we only have nonzero linear x values (the robot can only drive forwards or backwards) and nonzero angular z values (the robot can only rotate about its vertical axis). Note: $+x$ is forward, and $+z$ is counter clockwise rotation. For an example Twist message, see Fig. 8. In this case, the robot would go forwards at 0.4 m/s. Once the control node has received these values which are in meter/second and radians/second respectively, it converts them to the appropriate PWM value using the calculations described above. There is a further calibration which is multiplication by a factor determined empirically for the surface being driven on. In this case, a factor of 1.3 was used for the carpet in the Shanahan Center. This reflects the fact that driving on carpet requires more torque than driving on the calibration surface (e.g. on a lab bench or being held in the air).

The other half of the control node is the odometry calculation used in localizing the robot. The implementation makes use of the ROS tf package to determine poses and the transforms between the local frame and the global frame (odom_wheel). The mathematics of the transforms

```
rostopic pub \cmd\_vel geometry\_msgs/twist "Linear:
    X: 0.4
    Y: 0.0
    Z: 0.0
Angular:
    X: 0.0
    Y: 0.0
    Z: 0.0"
```

Fig. 8. Command used to instruct the robot to move forward.

are described above. Once the x and y positions as well as the orientation in the global frame are calculated they are broadcast using the tf package odom_broadcaster which publishes to the `/odom` topic. This topic can then be accessed by rviz and used to visualize the movement of the robot in a plane. `/odom` uses a nav_msg type: Odometry. These messages consist of a `Header` header, `string` child_frame_id, and `geometry_msgs`: `PoseWithCovariance` pose and `TwistWithCovariance` twist. The child_frame_id is base_link (the robot frame) and is the frame used to specify the twist in. The pose is specified in the given header_frame_id; odom_wheel (the global frame). This is calculated by the odometry functionality. `PoseWithCovariance` contains a `Pose` pose datamember which is made of two datamembers `Point` position and `Quaternion` orientation both of which are updated each time odom_pub is called. The `Point` is just `float_64` x,y and z values. The `Quaternion` is `float_64` x,y,z and w. The control node also published the range readings from the IR sensors using a custom `ir_sensor` message type, which has a single `float_32` datamember: `distance`.

V. Conclusions

Based on the work presented above, our robot is now able to

1) Carry out instructions to move in arbitrary linear combinations of translational x motion and rotational z motion, receiving those instructions from a ROS topic,
2) Use wheel odometry to accurately estimate its location and orientation in 2D space, publishing that data to a ROS topic, and
3) Use IR sensors to measure the range to obstacles in the $+x$, $+y$, and $-y$ directions, publishing that data to a ROS topic.

These capabilities lay the foundation for future work implementing closed-loop control, path planning, obstacle detection, and autonomous navigation.

References

[1] Y. Chang. E190AR. Class Lecture, Topic: "Locomotion and Odometry." Department of Engineering, Harvey Mudd College, Claremont, CA. Slides by C. Clark, D.J. Cappelleri, and M. Salman.

[2] SHARP Corporation, "Distance Measuring Sensor Unit," GP2Y0A02YK0F datasheet, Dec. 2006.